(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号
特開2001－184230
（P2001－184230A）
(43)公開日　平成13年７月６日(2001.7.6)

(51)Int.Cl.'	識別記号	ＦＩ	テーマコード(参考)
Ｇ０６Ｆ　11/28		Ｇ０６Ｆ　11/28	Ｌ　５Ｂ０４２
	３０５		３０５Ａ　５Ｂ０７６
9/445		9/06	４２０Ｊ

審査請求　未請求　請求項の数１　ＯＬ　（全 12 頁）

(21)出願番号　特願平11－370382

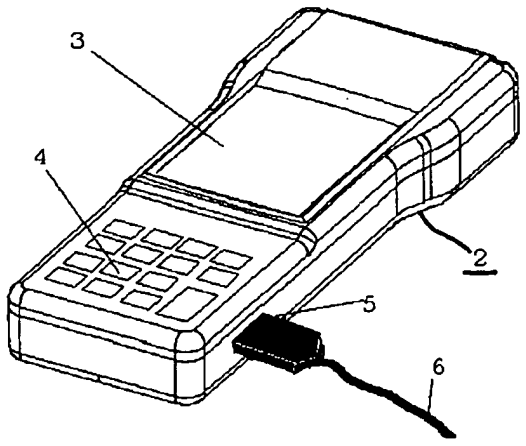
(22)出願日　平成11年12月27日(1999.12.27)

(71)出願人　000005511
ぺんてる株式会社
東京都中央区日本橋小網町７番２号
(72)発明者　中原　純
埼玉県草加市吉町４－１－８　ぺんてる株式会社草加工場内

Ｆターム(参考)　5B042 GA02 GA07 GA36 GC05 HH23
HH35 HH50 NN04 NN10
5B076 BB04 DB08 EB02

(54)【発明の名称】　ハンディターミナル用応用プログラムのデバッグシステム

(57)【要約】
【課題】　ハンディターミナルでシンボリックデバッグを簡単に行えるシステムを提供する。
【解決手段】　コンパイラが出力するオブジェクトコードを仮想機械語にし、仮想言語を解釈実行するインタプリタをハンディターミナルに搭載する。当該インタプリタにデバッグ機能を組み込み、ハンディターミナルとホストコンピュータが通信しながらシンボリックデバッグを行う。



1

【特許請求の範囲】

【請求項1】　ハンディターミナルでおこなわれるハンディターミナル向け日常業務プログラム（以下「応用プログラム」と略称する）のデバッグシステムであって、仮想機械語を出力する翻訳プログラムを用い、仮想機械語を解釈実行するプログラムをハンディターミナルで実行させ、デバッグ実行時、当該解釈実行プログラムがホストコンピュータと通信を行うことにより、シンボリックデバッグを行うこと特徴とするハンディターミナル用応用プログラムのデバッグシステム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、倉庫での在庫管理データの収集や、店頭での商品データを収集するハンディターミナルであり、このハンディターミナルの応用プログラムのデバッグシステムに関する。

【0002】

【従来の技術】ハンディターミナルの応用プログラムの開発ステップは以下である。まずホストコンピュータ（通常パーソナルコンピュータが利用される）でソースプログラムを作成し、翻訳プログラム（以下コンパイラと略称する）を利用し、ソースプログラムから実行形式プログラム（以下オブジェクトプログラムと略称する）を作成する。オブジェクトプログラムをホストコンピュータからハンディターミナルへダウンロードし、フラッシュROMなどの不揮発性メモリに焼き、当該プログラムをハンディターミナルで実行させ、プログラムの動作確認を行う。動作不正が起こった際は、ソースプログラムを見直し、変更を行い、再びコンパイル、ダウンロードを行い、ハンディターミナルで実行させ、動作確認を行う操作を繰り返していた。

【0003】

【発明が解決しようとする課題】しかし上記した従来技術では、ハンディターミナルのプログラム不正動作発見時に行うエラー解析と、プログラム修正作業（以下デバッグと略称する）は、多くのプログラム実行用資源（CPU、メモリ、ファイル、デバッグ用ユーティリティなど）を有しないハンディターミナルでは、シンボリックデバッガー（機械語やアドレス番地を用いず、コーディングしたソースプログラムコードを用いてデバッグを行うデバッグ用プログラム）など、高度なデバッガーが搭載できず、前記した様に、プログラムを試行錯誤で改変しながらエラー修正する方式しか提供されていなかった。そこでハンディターミナルの応用プログラムのデバッグにも、シンボリックデバッガーの搭載が求められていた。

【0004】

【課題を解決するための手段】本発明は上記問題に鑑みなされたものであり、倉庫での在庫管理データの収集や、店頭での商品データを収集するハンディターミナルにおいて、前記ハンディターミナルとホストコンピュータとの間でおこなわれる応用プログラム開発のデバッグ時、ハンディターミナルプログラムを特別に改変せずに、シンボリックデバッグを行えるシステムを提供するものである。

2

【0005】

【発明の実施の形態】ハンディターミナルの応用プログラム開発には、ハンディターミナルとホストコンピュータとをシリアル回線を用いて接続し、オブジェクトプログラムをダウンロードし、不揮発性メモリ（フラッシュROM等）を書き換える方式が採られる。つまり、ハンディターミナルの応用プログラムを開発する際は、必ずハンディターミナルとホストコンピュータとが準備されている。そこでデバッグ時、ハンディターミナルのオブジェクトプログラムを実行する際、ホストコンピュータと通信し、シンボリックデバッグ機能を実行させる仕組みを開発システムに組み込む。

【0006】ハンディターミナルのオブジェクトプログラムは仮想機械語で記述される。仮想機械語はハンディターミナルが具備する諸機能を効率よく実行する命令体系を有している。ホストコンピュータで動作するコンパイラは、当該仮想機械語を出力する様にする。仮想機械語はハンディターミナルで動作する解釈実行プログラム（以下インタプリタと略称する）で実行される。このインタプリタにデバッガーの機能を組み込む。

【作用】

【0007】ハンディターミナルで日常業務を記述したソースプログラムを準備する。本発明例のCOBOL言語コンパイラでは、ソースプログラムはCOBOL言語文法に則りプログラミングされる。ホストコンピュータのコンパイラは当該ソースプログラムをコンパイルし、仮想機械語のオブジェクトプログラムを出力する。

【0008】仮想機械語で記述されたオブジェクトプログラムは、ハンディターミナルにダウンロードされ、ハンディターミナルのフラッシュROMに焼かれる。

【0009】ハンディターミナルのインタプリタで応用プログラムを実行すると、ハンディターミナルのインタプリタは仮想機械語命令を逐一取り出し、解釈実行する。インタプリタは仮想機械語の実行動作を模擬する。

【0010】ハンディターミナルとホストコンピュータとをシリアルケーブルで接続し、特定キーを押しながらハンディターミナルの電源を投入すると、インタプリタの実行モードを変更させるようにする。デバッグモード時、インタプリタは取り出した仮想機械語のアドレスをホストコンピュータに送信し、応答を待つ。応答には、仮想機械語の実行指示や、アドレスのデータを読み出させるものを準備する。インタプリタは応答に従い各処理を行う。プログラマはホストコンピュータ上で、ハンディターミナルの実行中のプログラムアドレスや、メモリデータを見ることができる。また、実行中のプログラム

ソースや、変数の内容をホストコンピュータの表示装置で確認できる。

【0011】

【実施例】以下、本発明の詳細を図示実施例に基づいて説明する。図1は本発明を適用するホストコンピュータ1の外観斜視図である。ホストコンピュータ1では本発明の仮想機械語を生成するコンパイラや、シンボリックデバッガーが稼働するものである。図2は本発明を適用するハンディターミナル2の外観斜視図である。参照符号3はLCD表示部であり、収集したデータやハンディターミナル2から図示しない操作者へのメッセージを表示する部分である。参照符号4はテンキーや入力キーからなるキー入力部である。参照符号5は通信ポートであり、ホストコンピュータ1と通信を行う為にシリアル通信ケーブル6を接続する。

【0012】図3は、ハンディターミナル2の電気的ブロック図を示したものである。参照符号7は、ハンディターミナル2の制御手段である中央処理装置であり、書き換え可能なフラッシュROM8に格納されたプログラムに従って本発明に係わる処理を実行する。上記したキー入力部4を操作することにより、電源回路13を介してLCD表示部3に入力したデータを表示する。収集したデータをホストコンピュータ1に送信する場合には、通信インタフェース回路18を介し、通信ポート5に上記したシリアル通信ケーブル6を接続することにより、CPU7の指令によりデータの送信を行う。参照符号16はカレンダ回路であり、年月日時分秒情報を発生させる為に設けられており、年月日時分秒情報は図示しないバックアップ用のリチウムにより保護される。ブザー駆動回路12とLED駆動回路17もそれぞれCPU7の指令により、所定の動作を行う。参照符号11はプリンタでありCPU7の指令により、所定の動作を行う。なお、バーコードリーダ（図示せず）が接続され運用に供される。

【0013】図4はハンディターミナル2の本体に設けられているフラッシュROM8、RAM14に格納されているデータ並びを示したものである。参照符号19は日常業務プログラム21を解釈実行するインタプリタである。参照符号20は、当該ハンディターミナルに様々な動作を行わせる為の基本プログラムである。デバッグモードの切り替えの選択も当該プログラムの一部が行う。日常業務プログラム21はハンディターミナルの日常業務プログラムであり、仮想機械語で記述された実行用のオブジェクトプログラムである。当該コードは基本プログラム20でホストコンピュータからシリアルケーブル6を介して伝送され、焼き付けられたものである。日常業務プログラム21はハンディターミナル2の主たる業務、例えば、バーコードを入力し、当該コードを画面表示させ、数量などのキー入力された追加情報をワーク領域22を利用し一時保管したり、収集データ23へ蓄積するなどの処理が記述されている。なお、ワーク領域22には仮想機械語が利用するワークデータも置かれる。

【0014】図4の参照符号24はモードフラグである。当該フラグはハンディターミナル2のキー入力部4にある入力キーを押しながら電源投入すると、基本プログラム20にあるモード変更処理（図示せず）が呼び出され、通常モードが選択されると0が書き込まれ、デバッグモードが選択されると1が書き込まれる。

【0015】図5はハンディターミナル2で稼働する例題プログラムのリストを示す。図6、図7は図5のプログラムをコンパイルし、仮想機械語に翻訳し、作成された機械語をアセンブラ言語で併記したリストである。ソースステートメントと展開された機械語とが表示されている。当該リストを用いれば、機械語のアドレスから、展開されたソースステートメントがわかる。また、ソースプログラム上で記述された変数の機械語でのアドレスを知ることができる。ホストコンピュータ1はファイル形式で当該リストをコンピュータ内部に保持し、後の処理で利用できるようにしている。図8は本発明の仮想機械語の一覧表である。

【0016】図9はホストコンピュータ1で稼働するコンパイラ（図示せず）の動作フローを示している。以下当該フローに沿って説明する。ソースプログラムから仮想機械語命令を生成するには、まずソースプログラムの字句を分解し、字句の種類を予め定められた型に割り当てる字句解析処理を行う。（S1）例えば”ADDDAT1 TO DATAAREA.”というステートメントを字句解析すると図10の様に「予約語、変数、予約語、変数、予約語」の並びであると解析される。

【0017】次に分解された字句の並びを受け取り、構文解析処理を実施する。（S2）構文解析は、字句の型の並びの一覧表を内部に備え、どの構文に一致するか調べる。図11に構文解析処理が利用する型の一覧表（抜粋）を示す。本例の構文解析では”ADD DAT1 TO DATAAREA.”は、図11の参照符号29「ADD 変数 TO 変数」に一致すると解析される。

【0018】次にコード生成処理を実行する。（S3）当該処理は「ADD 変数 TO変数」を受け取り図12に示したような仮想機械語を生成する。

【0019】図15はハンディターミナル2で稼働するインタプリタ19の動作フローを示している。以下当該フローに沿って説明する。インタプリタが開始されると当該プログラムが利用するプログラムカウンタ（以下PCと略称する）とスタックポインタ（以下SPと略称する）を初期化する。オブジェクトプログラムの先頭アドレスとPCを加算した値が実行すべき命令アドレスになる。またSPはPCを待避するアドレスが格納される。（S4，S5）具体的に、PCは0が設定され、SPは

図４の２２内にワーク域が設定される。

【００２０】次に実行モードフラグ２４を参照しデバッグモードであるか否か判定する。（Ｓ６）値が０の時は通常モードなので、仮想機械語の実行処理ステップ（Ｓ１１）に進む。デバッグモード（値が１）の場合、インタプリタ１９は現在のＰＣをホストコンピュータ１に、図１２の送信データフォーマットに従い送信する。（Ｓ７）次にホストコンピュータからの応答データを受信する。（Ｓ８）応答データは図１３に示される４バイト長の電文である。次に受信した応答データを調べる。（Ｓ９）応答データの先頭文字が"Ｓ"の時、ステップ実行命令なので（Ｓ１１）へ進む。応答データの先頭文字が"Ｄ"の時、応答データのアドレスと長さを取り出し、指定されたメモリ番地のデータを指定長さ分読み出し、ホストコンピュータ１へ、図１２のダンプデータのフォーマットで送信する。（Ｓ１０）

【００２１】通常実行モード（デバッグモード以外）か、デバッグモード時に"Ｓ"コマンドを受け取ると、インタプリタは仮想機械語の実行を行う。まずＰＣの指すアドレスの仮想機械語を読み出す。（Ｓ１１）具体的には前記の仮想機械語の先頭アドレスにＰＣの値を加算したアドレスの内容を読み出せばよい。当該値は図７の一覧に示されるコードになっている。次に上位３ビットの命令タイプを割り出す。（Ｓ１２）次に仮想命令番号を割り出す。（Ｓ１３）命令タイプから次のＰＣを計算する。（Ｓ１４）具体的には（Ｓ１２）で得た値を右に５ビット論理シフトし、当該値を２倍し、仮想命令語サイズの１を加えた値を現在のＰＣに加算すればよい。次に命令語番号の処理を実行する。（Ｓ１５）次に、再び（Ｓ６）ステップへジャンプする。

【００２２】命令実行は処理番号から各々の処理を実行する。例として、図６の参照符号２５の仮想機械語命令（ＰＣが$(0000)_{16}$）は、$(40)_{16}$が仮想機械語命令であり、当該命令は図７の参照符号２６で、命令タイプは$(010)_{2}$になる。命令タイプを２倍し１を加えたもの（この場合５になる）が命令語長になり、現在のＰＣに当該値を加えたものを次のＰＣにする。

【００２３】図７の参照符号２６はアセンブラ表記では「ａｄｄ」で示され、当該命令は命令語に続く２つの２バイトコードを引数として、最初の引数が指すアドレスに格納された数値を２番目の引数が指すアドレスの数値へ加算する。同様にその他の処理命令も図７に示されるよう、命令語に続くデータを引数とし（引数の無いものもある）処理を実行する。

【００２４】図７の参照符号２７の「ａｃｃ」命令はバーコードを読み込む命令が含まれる。ハンディターミナル２のインタプリタ１９は当該命令実行時バーコードリーダー（図示せず）からバーコードを読み込む。また図７の参照符号２８「ｄｉｓｐ」命令は、文字列を指定されたライン／カラム位置に表示させるものである。ハンディターミナル２はＬＣＤ表示部３に文字列を表示する。

【００２５】図１５はホストコンピュータ１で実行されるシンボリックデバッグ用プログラム（以下デバッガーと略称する。）の処理フローである。以下当該フローに沿って説明する。事前準備としてハンディターミナル２とホストコンピュータ１をシリアルケーブル６で接続しておく。ハンディターミナル２でデバッグを行う際は、キーボード４の入力キー（右下の大きなキー）を押しながら電源スイッチ（図示せず）を投入する。ハンディターミナル２の基本プログラム２０内にあるモード変更処理（図示せず）が起動されたら、デバッグモードを選択しておく。

【００２６】ホストコンピュータ１のデバッガーはハンディターミナル２からの送信データ（図１２）を受信する。（Ｓ１６）受信したらＰＣの値を表示する。（Ｓ１７）次に受信したＰＣの値から、図６のアセンブラリストのソース行を探し出し画面表示する。（Ｓ１８）これは図６のＰＣ相対アドレスから、直前にあるソースステートメント（図６で左端が">>"で示される表示行）を表示すればよい。例えばハンディターミナル２からホストコンピュータ１への送信データが$(500000)_{16}$の場合、図６の"０００："で示される行がＰＣアドレスであり、その直前にある">>"のステートメント "ADD 1 TO COUNT." を表示すればよい。次にデバッガーはホストコンピュータ１のキーボードの入力を待つ。

（Ｓ１９）図示しない操作者のキー入力された文字を判定し、ステップ実行か否か調べる。（Ｓ２０）具体的には、キーボードよりの入力が"Ｓ"のとき、ステップ実行なので、図１３の実行指示応答をハンディターミナル２に送信する。（Ｓ２１）"Ｄ"の時は変数名をキー入力させる。（Ｓ２２）入力された変数名と一致するソースステートメントの変数名を探す。（Ｓ２３）次に変数のアドレスと長さを探す。（Ｓ２４、Ｓ２５）例えば"COUNT"という変数名が指定された場合、図６のソースステートメントから ">>77 COUNT PIC 9(004)." の行を探し出す。当該ステートメントの次行を見ると "000F:" となっており、当変数が０００Ｆ番地に割り当てられていることがわかる。さらにソースステートメントの "PIC9(004)" から、４バイト長の変数である点もわかる。デバッガーは当該情報から図１３のダンプ指示応答を作成し送信する。（Ｓ２６）当該例では（４４０００Ｆ０４）$_{16}$をハンディターミナル２に送信する。ハンディターミナル２は当該電文に応答し、ダンプ指示応答データをホストコンピュータ１へ送るので、当該電文を受信し表示する。（Ｓ２７）ハンディターミナルの応答は"COUNT"変数に格納された値である。図示しない操作者は変数の値を確認できる。その後、再びＰＣを受信するステップにジャンプする。

【００２７】以上の一連の動作で、ハンディターミナル

2のオブジェクトプログラムを実行させながら、対応するソースプログラムのステートメントや実際の変数の内容表示が行えるので、プログラムのデバッグ作業が容易になる。また、ホストコンピュータ1で、PC表示やソースステートメント表示を毎回行わず、指定された値のPCを受信するまで、実行指示応答を出し続ける処理をデバッガーに組み込むことが考えられる。本処理では指定ステートメントが実行されるまで、他の処理を連続実行する事になる。これはブレークポイント機能（指定実行命令を見つけたら停止する）になる。また、ステップ実行と同一変数のダンプ指示応答を毎回行い、ダンプデータの値が変化したときのみ、PCとソースステートメントの表示を行う機能の組み込みも考えられる。本処理では、指定変数が変化した命令の割り出しを可能にさせる機能となる。つまり、どのステートメントが変数の内容を書き換えたか調べられる。これらの高度なデバッグ機能も当該デバッガーでは簡単に実現できる。これらにより充実したシンボリックデバッグ機能を提供できる。

【0028】

【発明の効果】以上説明したように、ハンディターミナルの応用プログラム開発時、これまではハンディターミナルの非力さから、シンボリックデバッグの様な、ソースコードと実行プログラムとを対応させるデバッグ環境が無く、デバッグ操作が困難であったが、当該発明方式では、ハンディターミナル側のインタプリタや実行プログラムに負荷をかけずに、デバッグ環境を実現できる。また、シンボリックデバッグ用に特別な再コンパイルを行うことなく、ホストコンピュータとの連携で、ソースプログラムを確認しながらハンディターミナルの動作や変数の表示を確認することができる。これらによりデバッグ操作が容易になり、大幅な開発作業効率の増大が可能になる。

【図面の簡単な説明】

【図1】　本発明を適用するホストコンピュータの外観斜視図

【図2】　本発明を適用するハンディターミナルの外観斜視図

【図3】　ハンディターミナルの電気的ブロック図

【図4】　記憶装置内のデータの並び

【図5】　例題プログラム

【図6】　例題プログラムのコンパイル結果（仮想機械語）

【図7】　例題プログラムのコンパイル結果（仮想機械語）

【図8】　仮想機械語の一覧

【図9】　コンパイラ

【図10】　字句解析例

【図11】　構文一覧

【図12】　コード生成例

【図13】　送信データ

【図14】　応答データ

【図15】　インタプリタ

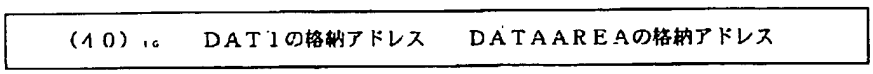
【図16】　デバッカー

【図17】　デバッカー操作例

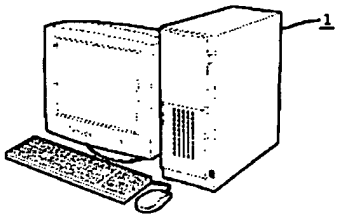
【符号の説明】

1	ホストコンピュータ
2	ハンディターミナル
3	LCD表示部
4	キー入力部
5	通信ポート
6	シリアル通信ケーブル
7	中央処理装置（CPU）
8	フラッシュROM
9	キー入力部
10	外部インタフェース
11	プリンタ
12	ブザー回路
13	電源回路
14	RAM
15	LCD表示部
16	カレンダ回路
17	LED駆動回路
18	通信インタフェース回路
19	インタプリタ
20	基本プログラム
21	業務プログラム
22	ワーク領域
23	収集データ
24	モードフラグ
25	最初の仮想機械語命令
26	add命令
27	disp命令
28	acc命令
29	add構文

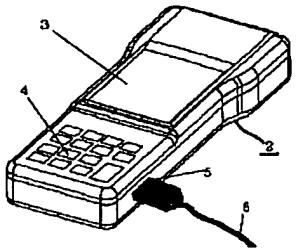
【図12】



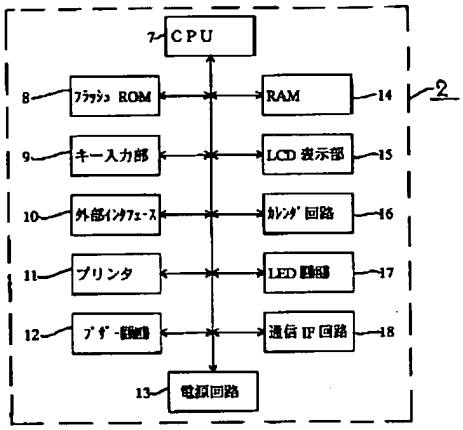
【図1】



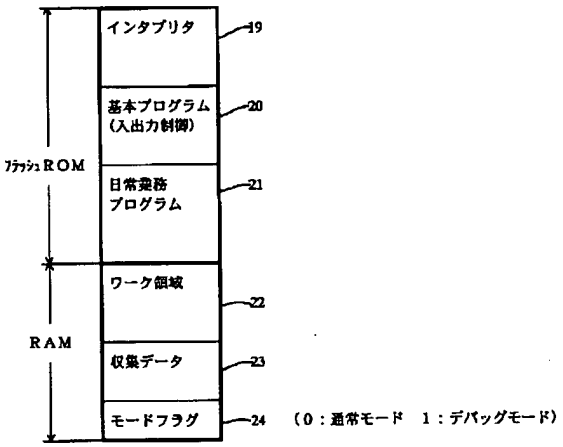
【図2】



【図3】



【図4】



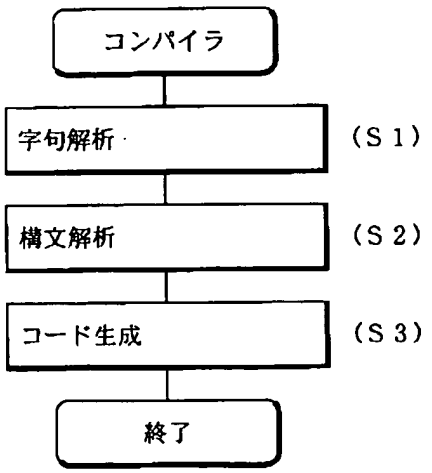
【図7】

```
(>>ソースステートメント)
(ワーク域アドレス：定数、ワーク)
>>FD      "TEST"  RECORD SIZE IS 10.
0000: 54 45 53 54                    @@0     DC      "TEST",00H
>>  01      BUF.
0005: 0000                           BUF     DS      0
>>    03      BUFFER          PIC     X(010).
0005: 000A                           BUFFER  DS      010
>>77      COUNT                 PIC     9(004).
000F: 30 30 30 30                    COUNT   DC      4 # "0",00H
>>77      BARBUF                PIC     X(064).
0014: 20 20 20 20 20 20 20 20        BARBUF  DC      64 # " ",00H
0055: 31                             @@1     DC      1,00H
0057: 31                             @@2     DC      1,00H
0059: 31                             @@3     DC      1,00H
005B: 31                             @@4     DC      1,00H
005D: 31 30 30 30                    @@5     DC      1000,00H
0062: 20 20 20 20 20 20 20 20        @@6     DC      "        ",00H
006B: 30                             @@7     DC      0,00H
006D: 31                             @@9     DC      1,00H
                                             END
```

【図5】

```
DATA                    DIVISION.
FILE                    SECTION.
FD      "TEST"  RECORD SIZE IS 10.
01      BUF.
  03      BUFFER          PIC     X(010).
WORKING-STORAGE         SECTION.
77      COUNT                   PIC     9(004).
77      BARBUF                  PIC     X(064).
PROCEDURE DIVISION.
*REIDAI PROGRAM FOR SAMPLE.
  ADD             1       TO      COUNT.
  SUBTRACT        1       FROM    COUNT.
  MULTIPLY        1       BY      COUNT.
  DIVIDE          1       INTO    COUNT.
  MOVE            1000    TO      COUNT.
  PERFORM         TEST.
  STOP    RUN.
TEST.
  DISPLAY "       "               AT      0000.
  IF      COUNT   EQUAL   0       THEN    OSIMAI.
  ACCEPT  BARBUF  FROM    BAR:    AT      0000.
  OPEN.
  GET     BUF     AT      COUNT.
  PUT     BUF     AT      COUNT.
  CLOSE.
  SUBTRACT        1       FROM    COUNT.
  GO      TO      TEST.
OSIMAI.
  EXIT.
  END
```

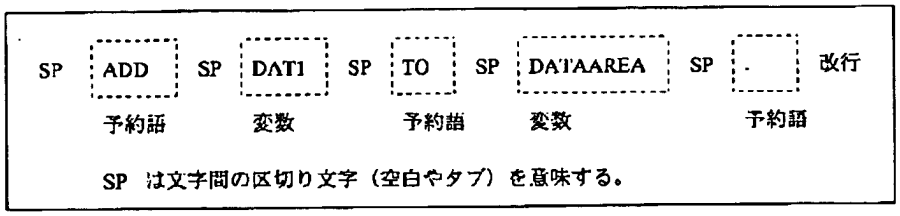
【図9】



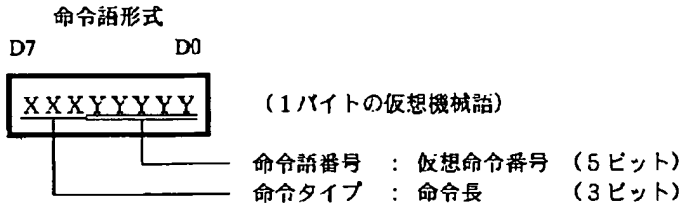
【図6】

```
（>>ソースステートメント）
（ＰＣ相対アドレス：仮想機械語コード）          （アセンブラ表記）
>>ADD            1       TO      COUNT.
0000: 40 0055 000F                              add     @@1,COUNT          25
>>SUBTRACT       1       FROM    COUNT.
0005: 41 0057 000F                              sub     @@2,COUNT
>>MULTIPLY       1       BY      COUNT.
000A: 42 0059 000F                              mul     @@3,COUNT
>>DIVIDE         1       INTO    COUNT.
000F: 43 005B 000F                              div     @@4,COUNT
>>MOVE           1000    TO      COUNT.
0014: 64 005D 000F 0004                         move    @@5,COUNT,4
>>PERFORM        TEST.
001B: 26 001F                                   pefm    TEST
>>STOP    RUN.
001E: 09                                        halt
>>TEST.
001F:                                   TEST
>>DISPLAY "              "              AT      0000.
001F: 4E 0062 00 00                             disp    @@6,0,0
>>IF      COUNT   EQUAL   0       THEN    OSIMAI.
0024: A5 000F 006B 000A 0052 002F               comp    COUNT,@@7,10,OSIMAI,_8
002F:                                   _8
>>ACCEPT  BARBUF  FROM    BAR:    AT      0000.
002F: 8D 0011 0000 0040 00 00                   acc     BARBUF,0,64,0,0
>>OPEN.
0038: 2A 0000                                   open    @@0
>>GET     BUF     AT      COUNT.
003B: 6B 0005 000F 000A                         get     BUF,COUNT,10
>>PUT     BUF     AT      COUNT.
0042: 6C 0005 000F 000A                         put     BUF,COUNT,10
>>CLOSE.
0049: 0F                                        cls
>>SUBTRACT       1       FROM    COUNT.
004A: 41 006D 000F                              sub     @@9,COUNT
>>GO      TO      TEST.
004F: 28 001F                                   jump    TEST
>>OSIMAI.
0052:                                   OSIMAI
>>EXIT.
0052: 07                                        exit
```

【図10】



【図8】



No	命令語	コード	引数1	引数2	引数3	引数4	引数5	
1	add	40	src ptr	dst ptr				26
2	sub	41	src ptr	dst ptr				
3	mul	42	src ptr	dst ptr				
4	div	43	src ptr	dst ptr				
5	move	64	src ptr	dst ptr	length			
6	comp	A5	comp1 ptr	comp2 ptr	condition	then ptr	else ptr	
7	pefm	26	ptr					
8	exit	07						
9	jump	28	ptr					
10	halt	09						
11	open	2A	file ptr					
12	get	6B	buf ptr	access ptr	length			
13	put	6C	buf ptr	access ptr	length			
14	acc	8D	dst ptr	dev	length	col	line	27
15	disp	4E	src ptr	col	line			28
16	cls	0F						

ptr　　：メモリのロケーションポインタ（2バイト）
length　：長さ　　（2バイト）
condition：比較演算子（2バイト）
col　　：カラム　　（1バイト）
line　　：ライン　　（1バイト）
dev　　：装置種別　（2バイト）

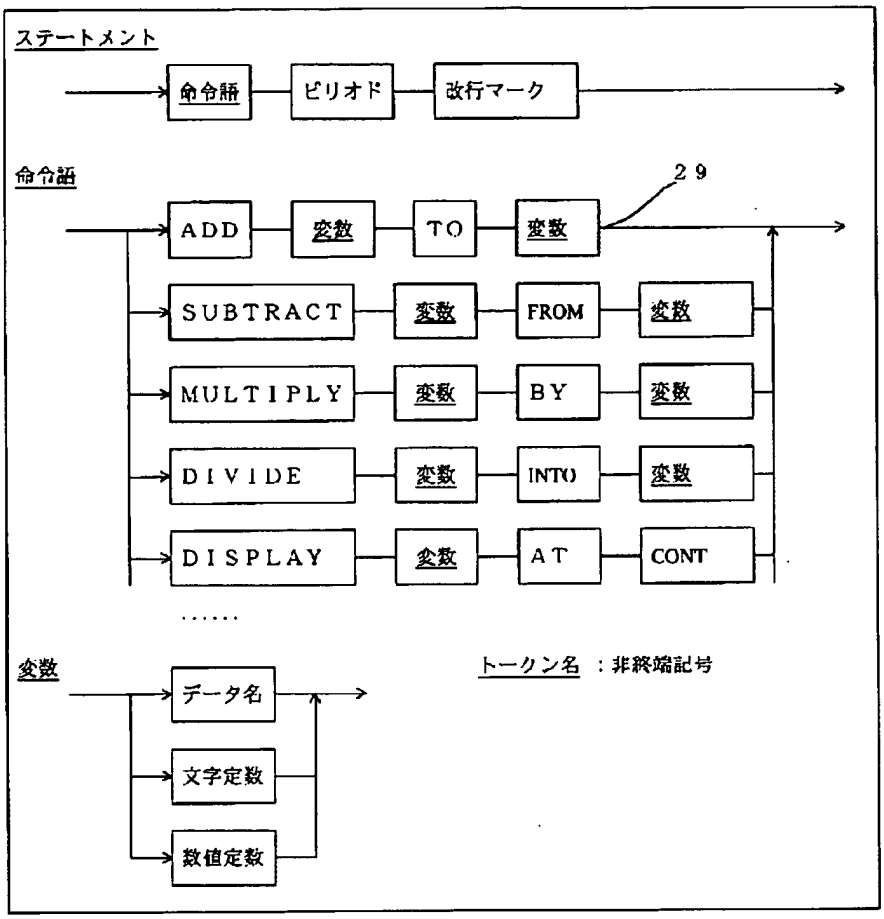
【図13】

"P" アドレス（2バイト）：　PCデータ
"D" データ...（nバイト）：　ダンプデータ

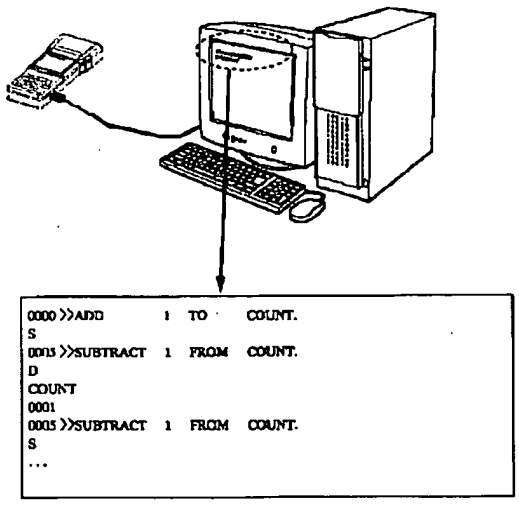
【図14】

"S000"　　：実行指示応答
"D" アドレス（2バイト）　長さ（1バイト）：ダンプ指示応答

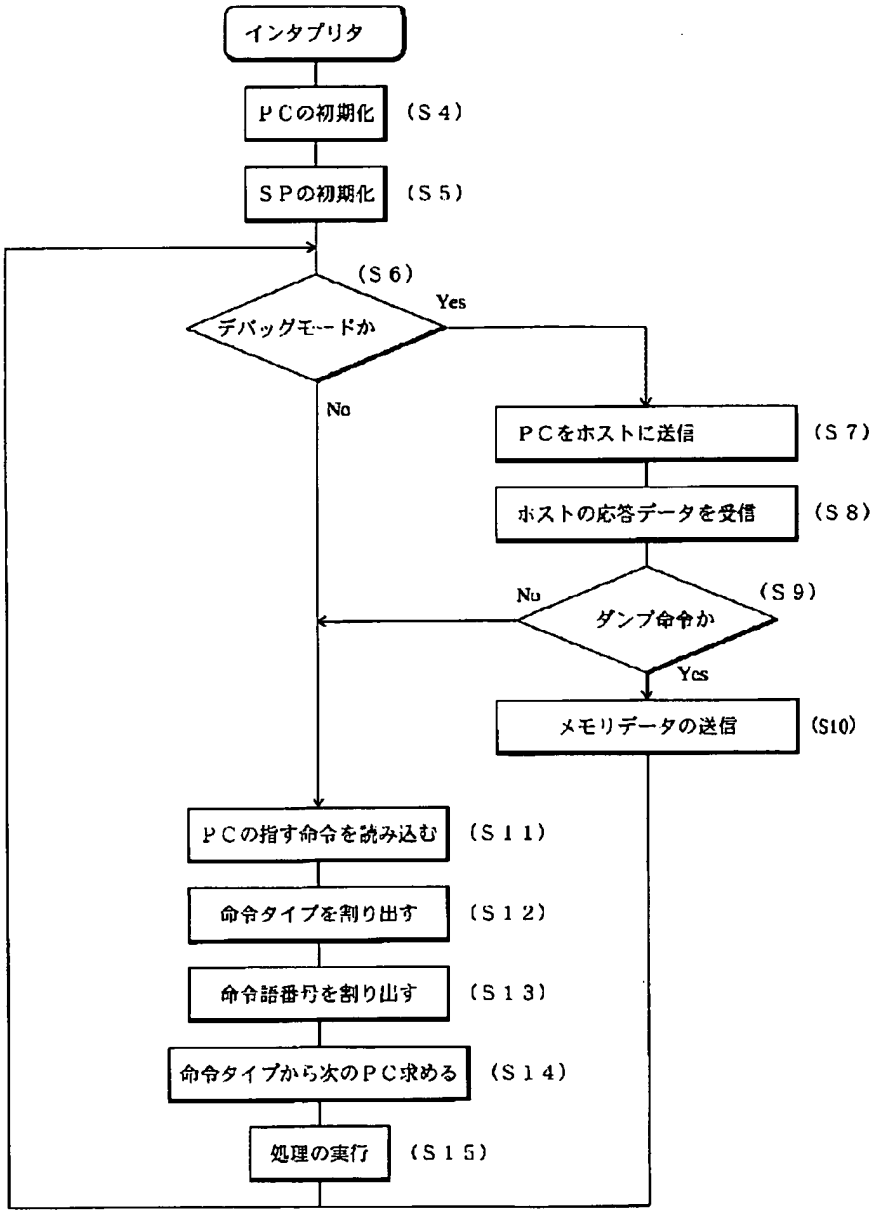
【図11】


ステートメント
命令語
ピリオド
改行マーク
命令語
29
ADD
変数
TO
変数
SUBTRACT
変数
FROM
変数
MULTIPLY
変数
BY
変数
DIVIDE
変数
INTO
変数
DISPLAY
変数
AT
CONT
.....
変数
データ名
文字定数
数値定数
トークン名 ：非終端記号

【図17】

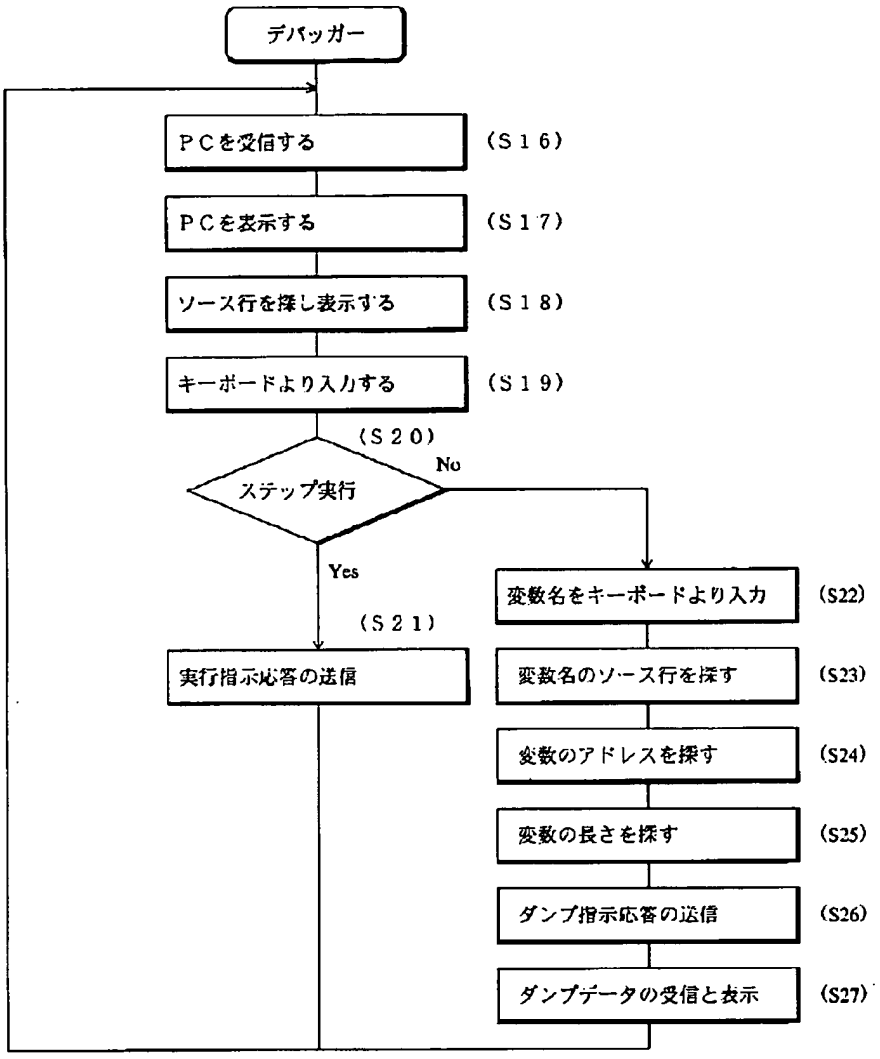

0000 >>ADD 1 TO COUNT.
S
0003 >>SUBTRACT 1 FROM COUNT.
D
COUNT
0001
0005 >>SUBTRACT 1 FROM COUNT.
S
...

【図15】



ＰＣ：プログラムカウンタ　　（実行すべき仮想機械語のアドレスポインタ）
ＳＰ：スタックポインタ　　（pefm,exit 用）

【図16】


デバッガー
PCを受信する
(S16)
PCを表示する
(S17)
ソース行を探し表示する
(S18)
キーボードより入力する
(S19)
(S20)
ステップ実行
No
Yes
(S21)
実行指示応答の送信
変数名をキーボードより入力
(S22)
変数名のソース行を探す
(S23)
変数のアドレスを探す
(S24)
変数の長さを探す
(S25)
ダンプ指示応答の送信
(S26)
ダンプデータの受信と表示
(S27)

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-184230

(43)Date of publication of application : 06.07.2001

(51)Int.Cl. G06F 11/28
G06F 9/445

(21)Application number : 11-370382

(22)Date of filing : 27.12.1999

(71)Applicant : PENTEL CORP

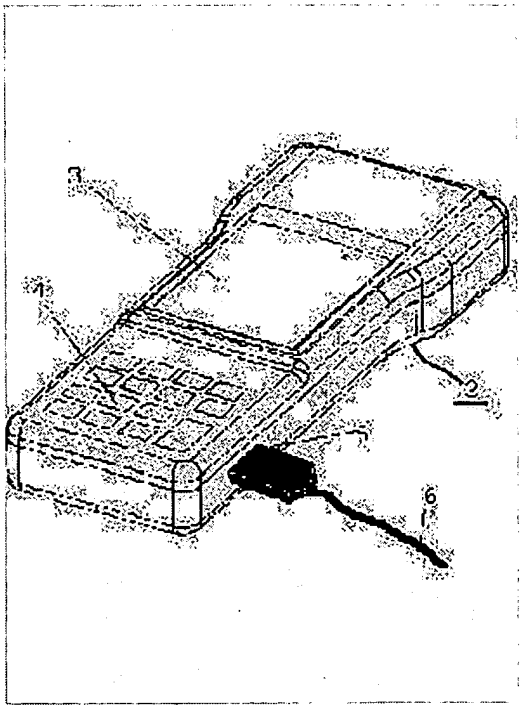
(72)Inventor : NAKAHARA JUN

(54) DEBUG SYSTEM FOR APPLICATION PROGRAM FOR HANDY TERMINAL

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a system capable of easily performing a symbolic debug by a handy terminal.

SOLUTION: An interpreter for converting object codes outputted by a complier into virtual machine language, and for interpreting virtual language is loaded on a handy terminal. A debug function is integrated into the interpreter, and symbolic debug is executed while communications between the handy terminal and a host computer are performed.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] This invention is a handy terminal which collects collection of the stock control data in a warehouse, and the goods data in a shop front, and relates to the debugging system of the application program of this handy terminal.
[0002]
[Description of the Prior Art] The development step of the application program of a handy terminal is the following. A source program is first created with a host computer (a personal computer is usually used), a compiler (it is called a compiler for short below) is used, and an execute-form program (it is called an object program for short below) is created from a source program. An object program is downloaded from a host computer to a handy terminal, it burns to nonvolatile memory, such as a flash ROM, the program concerned is performed by the handy terminal, and the check of a program of operation is performed. When injustice of operation happened, changed by having improved the source program, performed compile and download again, it was made to perform by the handy terminal, and actuation of performing a check of operation was repeated.
[0003]
[Problem(s) to be Solved by the Invention] However, the error analysis performed with the above-mentioned conventional technique at the time of program unjust actuation discovery of a handy terminal and a program correction activity (it is called debugging for short below) In the handy terminal which does not have many resources for program executions (CPU, memory, a file, utility for debugging, etc.) A symbolic debugger (program for debugging which debugs using the coded source program code not using an absolute language or an address address) etc., An advanced debugger could not be carried, but the above mentioned appearance was provided only with the method which carries out error correction, changing a program by trial-and-error. Then, debugging of the application program of a handy terminal was also asked for loading of a symbolic debugger.
[0004]
[Means for Solving the Problem] This invention is made in view of the above-mentioned problem, and offers the system which can perform symbolic debugging at the time of collection of the stock control data in a warehouse, and debugging of the application program development performed between said handy terminals and host computers in the handy terminal which collects the goods data in a shop front, without changing a handy terminal program specially.
[0005]
[Embodiment of the Invention] A handy terminal and a host computer are connected to application program development of a handy terminal using a serial circuit, an object program is downloaded to it, and the method which rewrites nonvolatile memory (flash ROM etc.) is taken. That is, in case the application program of a handy terminal is developed, the handy terminal and the host computer are surely prepared. Then, in case the object program of a handy terminal is performed at the time of debugging, it communicates with a host computer and the structure which performs a symbolic-debugging function is included in development system.
[0006] The object program of a handy terminal is described in a virtual machine word. The virtual machine word has the command structure which performs efficiently many functions in which a handy

terminal possesses. It is made for the compiler which operates with a host computer to output the virtual machine word concerned. A virtual machine word is performed by the interpretive program (it is called an interpreter for short below) which operates by the handy terminal. The function of a debugger is included in this interpreter.

[Function]

[0007] The source program which described the routine work by the handy terminal is prepared. In the COBOL language compiler of the example of this invention, a source program follows COBOL language syntax and is programmed. The compiler of a host computer compiles the source program concerned, and outputs the object program of a virtual machine word.

[0008] The object program described in the virtual machine word is downloaded to a handy terminal, and is burned by the flash ROM of a handy terminal.

[0009] If an application program is performed by the interpreter of a handy terminal, the interpreter of a handy terminal will take out a virtual machine word instruction in detail, and will carry out interpretation activation. An interpreter simulates activation actuation of a virtual machine word.

[0010] A handy terminal and a host computer are connected by the serial cable, and when the power source of a handy terminal is switched on pressing a specific key, it is made to make the execute mode of an interpreter change. At the time of a debug mode, an interpreter transmits the address of the taken-out virtual machine word to a host computer, and waits for a response. The thing to which activation directions of a virtual machine word and the data of the address are made to read is prepared for a response. An interpreter performs each processing according to a response. A programmer can see the program address and memory data under activation of a handy terminal on a host computer. Moreover, the program source under activation and the contents of the variable can be checked with the display of a host computer.

[0011]

[Example] Hereafter, the detail of this invention is explained based on an illustration example. Drawing 1 is the appearance perspective view of the host computer 1 which applies this invention. With a host computer 1, the compiler which generates the virtual machine word of this invention, and a symbolic debugger work. Drawing 2 is the appearance perspective view of the handy terminal 2 which applies this invention. A reference mark 3 is a LCD display and is a part which displays the message to the operator who illustrates neither from collected data nor a handy terminal 2. A reference mark 4 is the key input section which consists of a ten key or an input key. A reference mark 5 is a communication link port, and in order to communicate with a host computer 1, it connects the serial communication cable 6.

[0012] Drawing 3 shows the electric block diagram of a handy terminal 2. A reference mark 7 is a central processing unit which is the control means of a handy terminal 2, and performs processing concerning this invention according to the program stored in the rewritable flash ROM 8. By operating the above-mentioned key input section 4, the data inputted into the LCD display 3 through the power circuit 13 are displayed. In transmitting collected data to a host computer 1, it transmits data by the command of CPU7 by connecting the serial communication cable 6 described above in the communication link port 5 through the communication-interface circuit 18. A reference mark 16 is a calender circuit, and it is prepared in order to generate date time second information, and date time second information is protected by the lithium for backup which is not illustrated. By the command of CPU7, the buzzer drive circuit 12 and the LED drive circuit 17 also perform predetermined actuation, respectively. A reference mark 11 is a printer and performs predetermined actuation by the command of CPU7. In addition, a bar code reader (not shown) is connected and employment is presented.

[0013] Drawing 4 shows the flash ROM 8 prepared in the body of a handy terminal 2, and the data list stored in RAM14. A reference mark 19 is an interpreter which carries out interpretation activation of the routine-work program 21. A reference mark 20 is a basic program for making various actuation perform to the handy terminal concerned. A part of program concerned also performs selection of a change of a debug mode. The routine-work program 21 is a routine-work program of a handy terminal, and is an object program for activation described in the virtual machine word. By the basic program 20, from a host computer, the code concerned is transmitted through a serial cable 6, and can be burned. The routine-work program 21 inputs the main business of a handy terminal 2, for example, a bar code, a screen display of the code concerned is carried out, use the work-piece field 22, and the additional information which it keyed [quantity] is stored temporarily, or processing of accumulating to the

collection data 23 is described. In addition, the work-piece data which a virtual machine word uses are also put on the work-piece field 22.
[0014] The reference mark 24 of drawing 4 is a mode flag. The mode change processing (not shown) which is in a basic program 20 when it acts as powering on of the flag concerned, pushing the input key in the key input section 4 of a handy terminal 2 is called, if the normal mode is chosen, 0 will be written in, and 1 will be written in if a debug mode is chosen.
[0015] Drawing 5 shows the list of example programs which works by the handy terminal 2. Drawing 6 and drawing 7 are the lists which compiled the program of drawing 5 , translated into the virtual machine word, and wrote together the created absolute language with assembler language. Source statement and the developed absolute language are displayed. If the list concerned is used, the developed source statement is known from the address of an absolute language. Moreover, the address in the absolute language of the variable described on the source program can be known. A host computer 1 holds the list concerned inside a computer by file format, and enables it to use it by next processing. Drawing 8 is the chart of the virtual machine word of this invention.
[0016] Drawing 9 shows the flow of the compiler (not shown) which works with a host computer 1 of operation. In accordance with the flow concerned, it explains below. In order to generate a virtual machine word instruction from a source program, the token of a source program is parsed first and lexical-analysis processing assigned to the mold which was able to define the class of token beforehand is performed. (S1) For example, if the lexical of the statement "ADDDAT1 TO DATAAREA." is analyzed, it will be analyzed that it is the list of "reserved word, a variable, reserved word, a variable, and reserved word" like drawing 10 .
[0017] Next, reception and syntax-analysis processing are carried out for the list of the parsed token. (S2) It investigates whether syntax analysis is in agreement with which functor by equipping the interior with the chart of the type list of a token. The chart (extract) of the mold which syntax-analysis processing uses for drawing 11 is shown. By syntax analysis of this example "ADD DAT1 TO DATAAREA." will be analyzed if in agreement with the reference mark 29 of drawing 11 "an ADD variable TO variable."
[0018] Next, code generation processing is performed. (S3) The processing concerned generates a virtual machine word as showed the "ADD variable TO variable" to reception drawing 12 .
[0019] Drawing 15 shows the flow of the interpreter 19 which works by the handy terminal 2 of operation. In accordance with the flow concerned, it explains below. Initiation of an interpreter initializes the program counter (it calls for short Following PC) and stack pointer (it calls for short Following SP) which the program concerned uses. It becomes the start address of an object program, and the instruction address which the value adding PC should perform. Moreover, the address with which SP shunts PC is stored. (S4, S5) Concretely, as for PC, 0 is set up and, as for SP, a work-piece region is set up in 22 of drawing 4 .
[0020] Next, with reference to the execute mode flag 24, it judges whether it is a debug mode. (S6) Since it is the normal mode when a value is 0, it progresses to the executive operation step (S11) of a virtual machine word. When it is a debug mode (a value is 1), an interpreter 19 transmits the present PC to a host computer 1 according to the transmitting data format of drawing 12 . (S7) Next, the response data from a host computer are received. (S8) Response data are the wording of a telegram of the 4-byte length shown in drawing 13 . Next, the received response data are investigated. (S9) When the initial character of response data is "S", since it is a step run command (S11), it progresses. When the initial character of response data is "D", the address and die length of response data are taken out, the data of the specified memory address are read by assignment die length, and it transmits to a host computer 1 in a format of the discharge data of drawing 12 . (S10)
[0021] Usually, if the "S" command is received at the time of execute mode (except a debug mode), and a debug mode, an interpreter will perform a virtual machine word. The virtual machine word of the address which PC points out first is read. (S11) What is necessary is just to read the contents of the address which specifically added the value of PC to the start address of the aforementioned virtual machine word. The value concerned is the code shown in the list of drawing 7 . Next, the instruction type of a high order triplet is deduced. (S12) Next, a virtual command number is deduced. (S13) The following PC is calculated from an instruction type. (S14) being concrete (S12) -- what is necessary is to carry out the 5-bit logic shift of the acquired value to the right, to double the value concerned two, and

just to add the value which added 1 of virtual instruction word size to current PC Next, processing of an instruction word number is performed. (S15) Next, it jumps to a step again (S6).
[0022] An instruction execution performs each processing from a processing number. As an example, (40) 16 are a virtual machine word instruction, the instruction concerned is the reference mark 26 of drawing 7 , and an instruction type is set to 2 (010) by the virtual machine word instruction (PC is 16 (0000)) of the reference mark 25 of drawing 6 . What doubled the instruction type two and added 1 (set to 5 in this case) becomes instruction word length, and sets to the following PC what applied the value concerned to current PC.
[0023] With an assembler notation, the reference mark 26 of drawing 7 is shown by "add", and adds the instruction concerned to the numeric value of the address with which the 2nd argument points out the numeric value stored in the address which the first argument points out by making two 2-byte codes following instruction word into an argument. As other processing instructions are similarly shown in drawing 7 , the data following instruction word are made into an argument (there is also a thing without an argument), and processing is performed.
[0024] The instruction with which the "acc" instruction of the reference mark 27 of drawing 7 reads a bar code is included. The interpreter 19 of a handy terminal 2 reads a bar code from a bar code reader (not shown) at the time of the instruction execution concerned. Moreover, the reference mark 28 "disp" instruction of drawing 7 is displayed on Rhine/column position which had the character string specified. A handy terminal 2 displays a character string on the LCD display 3.
[0025] Drawing 15 is the processing flow of the program for symbolic debugging (it is called a debugger for short below.) performed with a host computer 1. In accordance with the flow concerned, it explains below. The host computer 1 is connected with the handy terminal 2 by the serial cable 6 as prior preparation. In case it debugs by the handy terminal 2, an electric power switch (not shown) is switched on pushing the input key (lower right big key) of a keyboard 4. The debug mode will be chosen if the mode change processing (not shown) in the basic program 20 of a handy terminal 2 is started.
[0026] The debugger of a host computer 1 receives the transmit data (drawing 12) from a handy terminal 2. (S16) The value of PC will be displayed if it receives. (S17) From the value of PC received next, a screen display of the source line of the assembler list of drawing 6 is discovered and carried out. (S18) This should just display the source statement (display line a left end is indicated to be by ">>" by drawing 6) in just before from PC relative address of drawing 6 . for example, -- a handy terminal -- two -- from -- a host computer -- one -- transmit data -- 16 (500000) -- it is -- a case -- drawing 6 -- " -- 0000 -- : -- " -- being shown -- having -- a line -- PC -- the address -- it is -- the -- just before -- it is -- " -- > -- > -- " -- a statement -- "ADD 1 TO COUNT." What is necessary is just to display. Next, a debugger waits for the input of the keyboard of a host computer 1. (S19) The alphabetic character which keyed the operator who does not illustrate is judged and it investigates whether it is stepwise execution. (S20) Since it is stepwise execution when the input from a keyboard is "S", specifically, the activation directions response of drawing 13 is transmitted to a handy terminal 2. (S21) The time of "D" makes a variable name key. (S22) The variable name of the source statement which is in agreement with the inputted variable name is looked for. (S23) Next, the address and die length of a variable are looked for. (S24, S25) When the variable name of "COUNT" is specified, it is from the source statement of drawing 6 . ">>77 COUNT PIC 9(004)." A line is discovered. If the next line of the statement concerned is seen "000F:" It turns out that it has become and this variable is assigned to 000F street. further -- source statement "PIC9 (004)" from -- the point which is the variable of 4-byte length is also known. From the information concerned, a debugger creates the discharge directions response of drawing 13 , and is transmitted. (S26) In the example concerned, 16 is transmitted to a handy terminal 2 (44000F04). Since a handy terminal 2 answers the wording of a telegram concerned and sends discharge directions response data to a host computer 1, it receives and displays the wording of a telegram concerned. (S27) The response of a handy terminal is the value stored in the "COUNT" variable. The operator who does not illustrate can check the value of a variable. Then, it jumps to the step which receives PC again.
[0027] Since the contents display of the statement of a corresponding source program or an actual variable can be performed in a series of above actuation, performing the object program of a handy terminal 2, the debugging activity of a program becomes easy. Moreover, with a host computer 1, neither PC display nor a source statement display is performed each time, but it is possible [it] to

include the processing which continues issuing an activation directions response in a debugger until it receives PC of the specified value. In this processing, continuation activation of other processings will be carried out until an assignment statement is performed. This becomes a break point function (it will stop, if an assignment run command is found). Moreover, only when the discharge directions response of the same variable as stepwise execution is performed each time and the value of discharge data changes, the inclusion of a function which performs the display of PC and source statement is also considered. In this processing, it becomes the function to make possible indexing of the instruction from which the assignment variable changed. That is, it is investigated which statement rewrote the contents of the variable. These advanced debugging functions are also easily realizable in the debugger concerned. The symbolic-debugging function which was substantial with these can be offered.
[0028]
[Effect of the Invention] Although there was no debugging environment on which the source code and executive program like symbolic debugging are made to run triggered by powerless [of a handy terminal] until now at the time of application program development of a handy terminal and debugging actuation was difficult as explained above, a debugging environment can be realized by the invention method concerned, without covering a load over the interpreter and executive program by the side of a handy terminal. Moreover, actuation of a handy terminal and the display of a variable can be checked by cooperation with a host computer, checking a source program, without performing recompilation special to symbolic debugging. Debugging actuation becomes easy by these and increase of large development working efficiency is attained.

[Translation done.]